Apex Triggers

Get started with Apex Triggers:

```
trigger ClosedOpportunityTrigger on Opportunity (before insert, before update) {
List<Task> newTask = new List <Task>();
   //Grab the Opportunity Id's from Opps that are Closed Won from the Context Variable
and store them in opp
   for(Opportunity opp : [SELECT Id FROM Opportunity
                   WHERE StageName = 'Closed Won' IN :Trigger.New]){
   //Create a Follow Up Task against Id's that are stored in the variable opp
   newTask.add(new Task(Subject = 'Follow Up Test Task',
                  Priority = 'High',
                  WhatId = opp.Id));
   //Insert new Tasks
                     {insert newTask;
     }
   }
}
```

Bulk Apex Triggers:

```
 @isTest
public class TestVerifyDate {
   @isTest static void Test_CheckDates_case1(){
      Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),
date.parse('01/05/2020'));
      System.assertEquals(date.parse('01/05/2020'), D);
   }
   @isTest static void Test_CheckDates_case2(){
      Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),
date.parse('05/05/2020'));
      System.assertEquals(date.parse('01/31/2020'), D);
   }
   @isTest static void Test_DateWithin30Days_case1(){
      Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('12/30/2019'));
      System.assertEquals(false, flag);
   }
   @isTest static void Test_DateWithin30Days_case2(){
      Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('02/02/2019'));
        System.assertEquals(false, flag);
    }
    @isTest static void Test_DateWithin30Days_case3(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('02/02/2020'));
        System.assertEquals(true, flag);
    }
    @isTest static void Test_SetEndOfMonthDate(){
        Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
    }
}
```

Apex Testing

Get Started with Apex Unit Tests:

```
 trigger RestrictContactByName on Contact (before insert, before update) {

	//check contacts prior to insert or update for invalid data
	For (Contact c : Trigger.New) {
		if(c.LastName == 'INVALIDNAME') {	//invalidname is invalid
			c.AddError('The Last Name "'+c.LastName+'" is not allowed for
DML');
		}

	}



}
```

Test Apex Triggers

```
 @isTest
public class TestRestrictContactByName {
    @isTest static void Test_insertupdateContact(){
        Contact cnt = new Contact();
        cnt.LastName = 'INVALIDNAME';
        Test.startTest();
        Database.SaveResult result = Database.insert(cnt, false);
        Test.stopTest();
        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',
result.getErrors()[0].getMessage());
    }
}
```

Create Test Data for Apex Testes

```
 public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer numcnt, string
lastname){
        List<Contact> contacts = new List<Contact>();
        for(Integer i=0;i<numcnt;i++){
            Contact cnt = new Contact(FirstName = 'Test '+i, LastName = lastname);
            contacts.add(cnt);
        }
        return contacts;
    }

}
```

Asynchronous Apex

Use Future Methods

```
public class AccountProcessor {
   @future
   public static void countContacts(List<Id> accountIds){

       List<Account> accountsToUpdate= new List<Account>();

       List<Account> accounts = [Select Id, Name, (Select Id from Contacts) from Account
Where Id in :accountIds];

       For (Account acc:accounts) {
          List<Contact> contactList = acc.Contacts;
          acc.Number_Of_Contacts__c= contactList.size();
          accountsToUpdate.add(acc);

       }
```

```
        update accountsToUpate;

    }
}
```

Use Batch Apexglobal class

```
LeadProcessor implements Database.Batchable<sObject> {
    global Integer count = 0;
    global Database.QueryLocator start(Database.BatchableContext bc){
        return Database.getQueryLocator('SELECT ID, LeadSource FROM Lead');
    }

    global void execute (Database.BatchableContext bc, List<Lead> L_list){
        List<lead> L_list_new = new List<lead>();
        for(lead L:L_list){
            L.leadsource = 'Dreamforce';
            L_list_new.add(L);
             count +=1;
        }
 update L_list_new;
    }
    global void finish(Database.BatchableContext bc){
        system.debug('count = ' + count );
    }
}
@isTest
public class LeadProcessorTest {
    @isTest
    public static void testit(){
        List<lead> L_list = new List<lead>();
        for(Integer i=0; i<200; i++){
            Lead L = new lead();
            L.LastName = 'name' + i;
            L.Company = 'Company';
            L.Status = 'Random Status';
            L_list.add(L);
        }
        insert L_list;
        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp);
        Test.stopTest();
    }

}
```

Control Processes with Queueable Apex

```
public class AddPrimaryContact implements Queueable{
    private Contact con;
    private String state;
    public AddPrimaryContact(Contact con, String state){
        this.con = con;
        this.state = state;
    }
    public void execute(QueueableContext context){
        List<Account> accounts = [Select Id, Name, (Select FirstName, LastName, Id from
contacts)
                                                    from Account where BillingState = :state Limit 200];
        List<Contact> primaryContacts = new List<Contact>();
        for(Account acc:accounts){
            Contact c = con.clone();
            c.AccountId = acc.Id;
primaryContacts.add(c);
        }
        if(primaryContacts.size() > 0){
            insert primaryContacts;
        }
    }

}
@isTest
public class AddPrimaryContactTest {
    static testmethod void testQueueable(){
        List<Account> testAccounts = new List<Account>();
        for(Integer i=0;i<50;i++){
            testAccounts.add(new Account(Name='Account '+i,BillingState='CA'));
        }
        for(Integer j=0;j<50;j++){
            testAccounts.add(new Account(Name='Account '+j,BillingState='NY'));
        }
        insert testAccounts;
        Contact testContact = new Contact(FirstName = 'John', LastName ='Doe');
        insert testContact;
        AddPrimaryContact addit = new addPrimaryContact(testContact, 'CA');
        Test.startTest();
system.enqueueJob(addit);
        Test.stopTest();
        System.assertEquals(50,[Select count()from Contact where accountId in (Select Id
from Account where BillingState='CA')]);
    }

}
```

Schedule jobs Using the Apex Schedule

```
 public class DailyLeadProcessor implements Schedulable{
    public void execute(SchedulableContext ctx){
        List<lead> leads = [Select Id From Lead Where LeadSource = NULL LIMIT 200];
        for(Lead l:leads){
            l.LeadSource = 'Dreamforce';
            update l;
        }
    }
}
Test.startTest();
        String jobId = System.schedule('DailyLeadProcessor', CRON_EXP, new
DailyLeadProcessor());
    }
}
```

Apex Integration Services

Apex REST Callouts

AnimalLocator.apxc (Class)

```
public class AnimalLocator {
    public static string getAnimalNameById(integer numSubmitted) {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + numSubmitted);
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        string replyName = 'None returned';
        if (response.getStatusCode() == 200) {
         replyName = response.getBody();
        }
            return replyName;
    }
}
```

AnimalLocatorTest.apxc

```
@IsTest
private class AnimalLocatorTest {
    @isTest static  void  testGetCallout() {
    // Set mock callout class
    Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
    // This causes a fake response to be sent
    // from the class that implements HttpCalloutMock.
    String animalname = AnimalLocator.getAnimalNameById(2);
    // Verify that the response received contains fake values
    String expectedValue = 'Charles H Bones Esquire';
    System.assertEquals(animalname, expectedValue);
    }
}
```

AnimalLocatorMock.apxc

```
@IsTest
global class AnimalLocatorMock implements HttpCalloutMock {
    //Implement this interface method
    global HTTPResponse respond(HTTPRequest request) {
        // Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('Charles H Bones Esquire');
        response.setStatusCode(200);
        return response;
    }
}
```

Apex Web Service

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager{
   @HttpGet
   global static Account getAccount(){
      RestRequest req = RestContext.request;
      String accId = req.requestURI.substringBetween('Accounts/', '/contacts');
      Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)
                  FROM Account WHERE Id = :accId];

      return acc;
   }
}



@IsTest
private class AccountManagerTest{
   @isTest static void testAccountManager(){
      Id recordId = getTestAccountId();
      // Set up a test request
      RestRequest request = new RestRequest();
      request.requestUri =
         'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts';
      request.httpMethod = 'GET';
      RestContext.request = request;

      // Call the method to test
      Account  acc = AccountManager.getAccount();

      // Verify results
      System.assert(acc != null);
   }

   private static Id getTestAccountId(){
      Account acc = new Account(Name = 'TestAcc2');
      Insert acc;
```

```
        Contact con = new Contact(LastName = 'TestCont2', AccountId = acc.Id);
        Insert con;

        return acc.Id;
    }
}

@IsTest
private class AccountManagerTest{
    @isTest static void testAccountManager(){
        Id recordId = getTestAccountId();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri =
            'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;

        // Call the method to test
        Account  acc = AccountManager.getAccount();

        // Verify results
        System.assert(acc != null);
    }

    private static Id getTestAccountId(){
        Account acc = new Account(Name = 'TestAcc2');
        Insert acc;

        Contact con = new Contact(LastName = 'TestCont2', AccountId = acc.Id);
        Insert con;

        return acc.Id;
    }
}
```

APEX SEVICES SUPERBADGE

```
public with sharing class MaintenanceRequestHelper {
public static void updateWorkOrders(List<Case> caseList) {
List<case> newCases = new List<Case>();
Map<String,Integer> result=getDueDate(caseList);
for(Case c : caseList){
if(c.status=='closed')
if(c.type=='Repair' || c.type=='Routine Maintenance'){
Case newCase = new Case();
newCase.Status='New';
newCase.Origin='web';
newCase.Type='Routine Maintenance';
newCase.Subject='Routine Maintenance of Vehicle';
newCase.Vehicle__c=c.Vehicle__c;
newCase.Equipment__c=c.Equipment__c;
newCase.Date_Reported__c=Date.today();
if(result.get(c.Id)!=null)
newCase.Date_Due__c=Date.today()+result.get(c.Id);
else
newCase.Date_Due__c=Date.today();
newCases.add(newCase);
}
}
insert newCases;
}
//
public static  Map<String,Integer> getDueDate(List<case> CaseIDs){
Map<String,Integer> result = new Map<String,Integer>();
Map<Id, case> caseKeys = new Map<Id, case> (CaseIDs);
List<AggregateResult> wpc=[select Maintenance_Request__r.ID
cID,min(Equipment__r.Maintenance_Cycle__c)cycle
from Work_Part__c where  Maintenance_Request__r.ID in :caseKeys.keySet() group by
Maintenance_Request__r.ID ];
for(AggregateResult res :wpc){
Integer addDays=0;
if(res.get('cycle')!=null)
addDays+=Integer.valueOf(res.get('cycle'));
```

```
result.put((String)res.get('cID'),addDays);
}
return result;
}
}

trigger MaintenanceRequest on Case (before update, after update) {
// ToDo: Call MaintenanceRequestHelper.updateWorkOrders
if(Trigger.isAfter)
MaintenanceRequestHelper.updateWorkOrders(Trigger.New);
}

public with sharing class WarehouseCalloutService {
private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
@future(callout=true)
public static void runWarehouseEquipmentSync() {
//ToDo: complete this method to make the callout (using @future) to the
//      REST endpoint and update equipment on hand.
HttpResponse response = getResponse();
if(response.getStatusCode() == 200)
{
List<Product2> results = getProductList(response); //get list of products from Http
callout response
if(results.size() >0)
upsert results Warehouse_SKU__c; //Upsert the products in your org based on the
external ID SKU
}
}
//Get the product list from the external link
public static List<Product2> getProductList(HttpResponse response)
{
List<Object> externalProducts = (List<Object>)
JSON.deserializeUntyped(response.getBody()); //desrialize the json response
List<Product2> newProducts = new List<Product2>();
for(Object p : externalProducts)
{
```

```
Map<String, Object> productMap = (Map<String, Object>) p;
Product2 pr = new Product2();
//Map the fields in the response to the appropriate fields in the Equipment object
pr.Replacement_Part__c = (Boolean)productMap.get('replacement');
pr.Cost__c = (Integer)productMap.get('cost');
pr.Current_Inventory__c = (Integer)productMap.get('quantity');
pr.Lifespan_Months__c = (Integer)productMap.get('lifespan') ;
pr.Maintenance_Cycle__c = (Integer)productMap.get('maintenanceperiod');
pr.Warehouse_SKU__c = (String)productMap.get('sku');
pr.ProductCode = (String)productMap.get('_id');
pr.Name = (String)productMap.get('name');
newProducts.add(pr);
}
return newProducts;
}
// Send Http GET request and receive Http response
public static HttpResponse getResponse() {
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint(WAREHOUSE_URL);
request.setMethod('GET');
HttpResponse response = http.send(request);
return response;
}
}
```